Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τεταρτη Εργαστηριακη Αναφορα
Λειτουργικα Συστηματα

Ομάδα 25
Αναστασία Χριστίνα Λίβα
03119029
Γεώργιος Μυστριώτης
03119065

Περιεχόμενα

Άσκηση 1

Ακολουθεί ο πηγαίος κώδικας για την πρώτη άσκηση:

```
/*
 * mmap.c
 *
 * Examining the virtual memory of processes.
 *
 * Operating Systems course, CSLab, ECE, NTUA
 *
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdint.h>
#include <signal.h>
#include <sys/wait.h>
#include <inttypes.h>
#include "help.h"

#define RED     "\033[31m"
#define RESET   "\033[0m"


char *heap_private_buf;
char *heap_shared_buf;

char *file_shared_buf;

uint64_t buffer_size;
uint64_t vag;
uint64_t va;
uint64_t vat;
```

```
/*
 * Child process' entry point.
 */
void child(void)
{
        uint64_t pa;
        int i;
        /*
         * Step 7 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete
         child's part of Step 7.
         */

        show_maps();
        /*
         * Step 8 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to
         complete child's part of Step 8.
         */

        pa=get_physical_address(vag);
        printf("child pa=%p\n", pa);

        /*
         * Step 9 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to
         complete child's part of Step 9.
         */
```

```
int* ptr;
ptr=vag;
int N=buffer_size/sizeof(int);
for (i=0; i<N; i++) {
        ptr[i]=1;
}

pa=get_physical_address(vag);
printf("child pa=%p\n", pa);

/*
 * Step 10 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to
 complete child's part of Step 10.
 */

ptr=va;
N=buffer_size/sizeof(int);
for (i=0; i<N; i++) {
        ptr[i]=1;
}
pa=get_physical_address(va);
printf("child pa=%p\n", pa);
/*
 * Step 11 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to
 complete child's part of Step 11.       */



mprotect(va, buffer_size, PROT_READ);
show_maps();
```

```
        /*
         * Step 12 - Child
         */
        /*
         * TODO: Write your code here to
         complete child's part of Step 12.
         */
        munmap(vag, buffer_size);
        munmap(va, buffer_size);
}

/*
 * Parent process' entry point.
 */
void parent(pid_t child_pid)
{
        uint64_t pa;
        int status;
        int i;

        /* Wait for the child to raise its first SIGSTOP. */
        if (-1 == waitpid(child_pid, &status, WUNTRACED))
                die("waitpid");

        /*
         * Step 7: Print parent's and child's
         maps. What do you see?
         * Step 7 - Parent
         */
        printf(RED "\nStep 7: Print parent's
        and child's map.\n" RESET);
        press_enter();

        /*
         * TODO: Write your code here to complete
         parent's part of Step 7.
         */
        show_maps();
        if (-1 == kill(child_pid, SIGCONT))
                die("kill");
```

```
if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


/*
 * Step 8: Get the physical memory
 address for heap_private_buf.
 * Step 8 - Parent
 */
printf(RED "\nStep 8: Find the
        physical address of the private heap "
        "buffer (main) for both the parent
        and the child.\n" RESET);
press_enter();

/*
 * TODO: Write your code here to complete
 parent's part of Step 8.
 */
pa=get_physical_address(vag);
printf("parent pa=%p\n",pa);
if (-1 == kill(child_pid, SIGCONT))
        die("kill");
if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


/*
 * Step 9: Write to heap_private_buf. What happened?
 * Step 9 - Parent
 */
printf(RED "\nStep 9: Write to the private
buffer from the child and "
        "repeat step 8. What happened?\n" RESET);
press_enter();

/*
 * TODO: Write your code here to
 complete parent's part of Step 9.
 */
```

```
pa=get_physical_address(vag);
printf("parent pa=%p\n",pa);

if (-1 == kill(child_pid, SIGCONT))
        die("kill");
if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


/*
 * Step 10: Get the physical memory
 address for heap_shared_buf.
 * Step 10 - Parent
 */
printf(RED "\nStep 10: Write to the shared heap buffer
(main) from "
        "child and get the physical
        address for both the parent and "
        "the child. What happened?\n" RESET);
press_enter();

/*
 * TODO: Write your code here to
 complete parent's part of Step 10.
 */
int* ptr;
ptr=va;
int N=buffer_size/sizeof(int);
for (i=0; i<N; i++) {
        ptr[i]=1;
}

pa=get_physical_address(va);
printf("parent pa=%p\n",pa);
if (-1 == kill(child_pid, SIGCONT))
        die("kill");
if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");
```

```
        /*
         * Step 11: Disable writing on the
         shared buffer for the child
         * (hint: mprotect(2)).
         * Step 11 - Parent
         */
        printf(RED "\nStep 11: Disable writing on the
                shared buffer for the "
                "child. Verify through the maps
                for the parent and the "
                "child.\n" RESET);
        press_enter();
        show_maps();
        /*
         * TODO: Write your code here to complete
         parent's part of Step 11.
         */

        if (-1 == kill(child_pid, SIGCONT))
                die("kill");
        if (-1 == waitpid(child_pid, &status, 0))
                die("waitpid");


        /*
         * Step 12: Free all buffers for parent and child.
         * Step 12 - Parent
         */

        /*
         * TODO: Write your code here to complete
         parent's part of Step 12.
         */
        munmap(vag, buffer_size);
        munmap(va, buffer_size);
}

int main(void)
{
```

```
pid_t mypid, p;
int fd = -1;
uint64_t pa;
int i;
mypid = getpid();
buffer_size = 1 * get_page_size();
/*
 * Step 1: Print the virtual address
 space layout of this process.
 */
printf(RED "\nStep 1: Print the
        virtual address space map of this "
        "process [%d].\n" RESET, mypid);
press_enter();
show_maps();
/*
 * TODO: Write your code here to complete Step 1.
 */


/*
 * Step 2: Use mmap to allocate a buffer
 of 1 page and print the map
 * again. Store buffer in heap_private_buf.
 */
printf(RED "\nStep 2: Use mmap(2) to allocate
        a private buffer of "
        "size equal to 1 page and print
        the VM map again.\n" RESET);
press_enter();
vag=mmap(NULL, buffer_size, PROT_WRITE|PROT_READ,
MAP_PRIVATE|MAP_ANONYMOUS, fd, 0);
show_maps();
show_va_info(vag);
/*
 * TODO: Write your code here
 to complete Step 2.
 */
```

```
/*
 * Step 3: Find the physical
 address of the first page of your buffer
 * in main memory. What do you see?
 */
printf(RED "\nStep 3: Find and print
        the physical address of the "
        "buffer in main memory.
        What do you see?\n" RESET);
press_enter();
/*
 * TODO: Write your code here to complete Step 3.
 */

get_physical_address(vag);

/*
 * Step 4: Write zeros to the buffer and repeat Step 3.
 */
printf(RED "\nStep 4: Initialize your
        buffer with zeros and repeat "
        "Step 3. What happened?\n" RESET);
press_enter();
/*
 * TODO: Write your code here to complete Step 4.
 */
int N=buffer_size/sizeof(int);
int *ptr=vag;

for (i=0; i<N; i++) {
        ptr[i]=0;
}
show_va_info(ptr);
pa=get_physical_address(vag);
printf("%p\n", pa);
/*
 * Step 5: Use mmap(2) to map file.txt
 (memory-mapped files) and print
 * its content. Use file_shared_buf.
 */
```

```
printf(RED "\nStep 5: Use mmap(2) to
        read and print file.txt. Print "
        "the new mapping information
        that has been created.\n" RESET);
press_enter();
/*
 * TODO: Write your code here to complete Step 5.
 */
fd=open("file.txt", O_RDONLY);
struct stat sb;
fstat(fd, &sb);
printf("fd=%d\n", fd);
vat=mmap(NULL, buffer_size, PROT_WRITE|PROT_READ,
MAP_PRIVATE, fd, 0);
char* f = vat;
i=0;
while(i<sb.st_size) {
        printf("%c", *f);
        f++;
        i++;}
show_maps();
show_va_info(vat);
/*
 * Step 6: Use mmap(2) to allocate a shared
 buffer of 1 page. Use
 * heap_shared_buf.
 */
printf(RED "\nStep 6: Use mmap(2) to allocate a
        shared buffer of size "
        "equal to 1 page. Initialize the
        buffer and print the new "
        "mapping information that
        has been created.\n" RESET);
press_enter();
/*
 * TODO: Write your code here to complete Step 6.
 */
va=mmap(NULL, buffer_size, PROT_WRITE|PROT_READ,
```

```
        MAP_SHARED|MAP_ANONYMOUS, -1, 0);
        show_maps();
        show_va_info(va);



        printf("fd=%d\n", fd);
        p = fork();
        if (p < 0)
                die("fork");
        if (p == 0) {
                child();
                return 0;
        }

        parent(p);
        munmap(vat, buffer_size);
        if (-1 == close(fd))
                perror("close");
        return 0;
}
```

Ερώτηση 1

Τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας χρησιμοποιώντας την εντολή `show_maps()` και λαμβάνουμε το εξής αποτέλεσμα:

```
Step 1: Print the virtual address space map of this process [16026].


Virtual Memory Map of process [16026]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00b8e000-00baf000 rw-p 00000000 00:00 0                                  [heap]
7f0f452aa000-7f0f4544b000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4544b000-7f0f4564b000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4564b000-7f0f4564f000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4564f000-7f0f45651000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f45651000-7f0f45655000 rw-p 00000000 00:00 0
7f0f45655000-7f0f4566d000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4566d000-7f0f4586c000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586c000-7f0f4586d000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586d000-7f0f4586e000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586e000-7f0f45872000 rw-p 00000000 00:00 0
7f0f45872000-7f0f45893000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a85000-7f0f45a88000 rw-p 00000000 00:00 0
7f0f45a8d000-7f0f45a92000 rw-p 00000000 00:00 0
7f0f45a92000-7f0f45a93000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a93000-7f0f45a94000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a94000-7f0f45a95000 rw-p 00000000 00:00 0
7ffe0653e000-7ffe0655f000 rw-p 00000000 00:00 0                          [stack]
7ffe065ba000-7ffe065bd000 r--p 00000000 00:00 0                          [vvar]
7ffe065bd000-7ffe065bf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Ερώτηση 2

Χρησιμοποιώντας την κλήση συστήματος `mmap()` δεσμεύουμε `buffer` μεγέθους μιας σελίδας και τυπώνουμε ξανά το χάρτη μνήμης

```
Step 2: Use mmap(2) to allocate a private buffer of size equal to 1 page and print the VM map again.


Virtual Memory Map of process [16026]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00b8e000-00baf000 rw-p 00000000 00:00 0                                  [heap]
7f0f452aa000-7f0f4544b000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4544b000-7f0f4564b000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4564b000-7f0f4564f000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f4564f000-7f0f45651000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f0f45651000-7f0f45655000 rw-p 00000000 00:00 0
7f0f45655000-7f0f4566d000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4566d000-7f0f4586c000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586c000-7f0f4586d000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586d000-7f0f4586e000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f0f4586e000-7f0f45872000 rw-p 00000000 00:00 0
7f0f45872000-7f0f45893000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a85000-7f0f45a88000 rw-p 00000000 00:00 0
7f0f45a8c000-7f0f45a92000 rw-p 00000000 00:00 0
7f0f45a92000-7f0f45a93000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a93000-7f0f45a94000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f0f45a94000-7f0f45a95000 rw-p 00000000 00:00 0
7ffe0653e000-7ffe0655f000 rw-p 00000000 00:00 0                          [stack]
7ffe065ba000-7ffe065bd000 r--p 00000000 00:00 0                          [vvar]
7ffe065bd000-7ffe065bf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f0f45a8c000-7f0f45a92000 rw-p 00000000 00:00 0
```

Ο `buffer` που δεσμεύτηκε φαίνεται στην όγδοη σειρά από το τέλος.

Ερώτηση 3

Προσπαθώντας να τυπώσουμε την φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η φυσική διεύθυνση του `buffer` βλέπουμε πως δεν έχει δεσμευτεί φυσική μνήμη. Αυτό είναι απόλυτα λογικό καθώς δεν έχουμε γράψει ακόμα κάτι σε αυτή τη θέση μνήμης, συνεπώς δεν έχει νόημα η δέσμευσή της (`on demand paging`).

```
Step 3: Find and print the physical address of the buffer in main memory. What do you see?

VA[0x7f5e8cee0000] is not mapped; no physical memory allocated.
```

Ερώτηση 4

Γεμίζοντας με μηδενικά το `buffer` γράφουμε στη θέση μνήμης, οπότε πλέον έχει δεσμευτεί φυσική μνήμη.

```
Step 4: Initialize your buffer with zeros and repeat Step 3. What happened?

7f5e8cedf000-7f5e8cee5000 rw-p 00000000 00:00 0
0xafe80000
```

Ερώτηση 5

Δημιουργούμε έναν καινούριο `private buffer` τον οποίο φτιάχνουμε χωρίς το `flag MAP_ANONYMOUS` ώστε να έχουμε τη δυνατότητα να αλληλεπιδρούμε με αρχεία. Δίνουμε ως όρισμα το `file descriptor` του αρχείου.

```
Hello everyone!


Virtual Memory Map of process [16184]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
01975000-01996000 rw-p 00000000 00:00 0                                  [heap]
7f54c2361000-7f54c2502000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f54c2502000-7f54c2702000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f54c2702000-7f54c2706000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f54c2706000-7f54c2708000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f54c2708000-7f54c270c000 rw-p 00000000 00:00 0
7f54c270c000-7f54c2724000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f54c2724000-7f54c2923000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f54c2923000-7f54c2924000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f54c2924000-7f54c2925000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f54c2925000-7f54c2929000 rw-p 00000000 00:00 0
7f54c2929000-7f54c294a000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f54c2b3c000-7f54c2b3f000 rw-p 00000000 00:00 0
7f54c2b42000-7f54c2b43000 rw-p 00000000 00:00 0
7f54c2b43000-7f54c2b44000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7f54c2b44000-7f54c2b49000 rw-p 00000000 00:00 0
7f54c2b49000-7f54c2b4a000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f54c2b4a000-7f54c2b4b000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f54c2b4b000-7f54c2b4c000 rw-p 00000000 00:00 0
7fff8e6bf000-7fff8e6e0000 rw-p 00000000 00:00 0                          [stack]
7fff8e6f2000-7fff8e6f5000 r--p 00000000 00:00 0                          [vvar]
7fff8e6f5000-7fff8e6f7000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------


7f54c2b43000-7f54c2b44000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
```

Έτσι μπορούμε να απεικονίσουμε το αρχείο στη μνήμη και να τυπώσουμε το περιεχόμενό του.

Ερώτηση 6

Δημιουργούμε έναν `shared buffer` και τον απεικονίζουμε στη μνήμη.

```
Virtual Memory Map of process [16593]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
0166a000-0168b000 rw-p 00000000 00:00 0                                  [heap]
7fa0c7374000-7fa0c7515000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7515000-7fa0c7715000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7715000-7fa0c7719000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7719000-7fa0c771b000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c771b000-7fa0c771f000 rw-p 00000000 00:00 0
7fa0c771f000-7fa0c7737000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7737000-7fa0c7936000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7936000-7fa0c7937000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7937000-7fa0c7938000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7938000-7fa0c793c000 rw-p 00000000 00:00 0
7fa0c793c000-7fa0c795d000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b4f000-7fa0c7b52000 rw-p 00000000 00:00 0
7fa0c7b54000-7fa0c7b55000 rw-p 00000000 00:00 0
7fa0c7b55000-7fa0c7b56000 rw-s 00000000 00:04 3994744                    /dev/zero (deleted)
7fa0c7b56000-7fa0c7b57000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7fa0c7b57000-7fa0c7b5c000 rw-p 00000000 00:00 0
7fa0c7b5c000-7fa0c7b5d000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5d000-7fa0c7b5e000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5e000-7fa0c7b5f000 rw-p 00000000 00:00 0
7ffedd3e7000-7ffedd408000 rw-p 00000000 00:00 0                          [stack]
7ffedd464000-7ffedd467000 r--p 00000000 00:00 0                          [vvar]
7ffedd467000-7ffedd469000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7fa0c7b55000-7fa0c7b56000 rw-s 00000000 00:04 3994744                    /dev/zero (deleted)
```

Το βλέπουμε στη θέση 10 από το τέλος.

Ερώτηση 7

Ακολουθεί ο χάρτης μνήμης για τη διεργασία πατέρα:

```
Virtual Memory Map of process [16593]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
0166a000-0168b000 rw-p 00000000 00:00 0                                  [heap]
7fa0c7374000-7fa0c7515000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7515000-7fa0c7715000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7715000-7fa0c7719000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7719000-7fa0c771b000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c771b000-7fa0c771f000 rw-p 00000000 00:00 0
7fa0c771f000-7fa0c7737000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7737000-7fa0c7936000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7936000-7fa0c7937000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7937000-7fa0c7938000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7938000-7fa0c793c000 rw-p 00000000 00:00 0
7fa0c793c000-7fa0c795d000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b4f000-7fa0c7b52000 rw-p 00000000 00:00 0
7fa0c7b54000-7fa0c7b55000 rw-p 00000000 00:00 0
7fa0c7b55000-7fa0c7b56000 rw-s 00000000 00:04 3994744                    /dev/zero (deleted)
7fa0c7b56000-7fa0c7b57000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7fa0c7b57000-7fa0c7b5c000 rw-p 00000000 00:00 0
7fa0c7b5c000-7fa0c7b5d000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5d000-7fa0c7b5e000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5e000-7fa0c7b5f000 rw-p 00000000 00:00 0
7ffedd3e7000-7ffedd408000 rw-p 00000000 00:00 0                          [stack]
7ffedd464000-7ffedd467000 r--p 00000000 00:00 0                          [vvar]
7ffedd467000-7ffedd469000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Ακολουθεί ο χάρτης μνήμης για τη διεργασία παιδί:

```
Virtual Memory Map of process [16595]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
0166a000-0168b000 rw-p 00000000 00:00 0                                  [heap]
7fa0c7374000-7fa0c7515000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7515000-7fa0c7715000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7715000-7fa0c7719000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c7719000-7fa0c771b000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fa0c771b000-7fa0c771f000 rw-p 00000000 00:00 0
7fa0c771f000-7fa0c7737000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7737000-7fa0c7936000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7936000-7fa0c7937000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7937000-7fa0c7938000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7fa0c7938000-7fa0c793c000 rw-p 00000000 00:00 0
7fa0c793c000-7fa0c795d000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b4f000-7fa0c7b52000 rw-p 00000000 00:00 0
7fa0c7b54000-7fa0c7b55000 rw-p 00000000 00:00 0
7fa0c7b55000-7fa0c7b56000 rw-s 00000000 00:04 3994744                    /dev/zero (deleted)
7fa0c7b56000-7fa0c7b57000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7fa0c7b57000-7fa0c7b5c000 rw-p 00000000 00:00 0
7fa0c7b5c000-7fa0c7b5d000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5d000-7fa0c7b5e000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fa0c7b5e000-7fa0c7b5f000 rw-p 00000000 00:00 0
7ffedd3e7000-7ffedd408000 rw-p 00000000 00:00 0                          [stack]
7ffedd464000-7ffedd467000 r--p 00000000 00:00 0                          [vvar]
7ffedd467000-7ffedd469000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Παρατηρούμε πως οι χάρτες μνήμης για τις διεργασίες γονιό-παιδί είναι ίδιοι καθώς μόλις έγινε το `fork()`.

Ερώτηση 8

```
Step 8: Find the physical address of the private heap buffer (main) for both the parent and the child.

parent pa=0x8dba2000
child pa=0x8dba2000
```

Τυπώνοντας τις φυσικές θέσεις μνήμης βλέπουμε πως είναι ίδιες. Όταν γίνεται ένα `fork()` ο χάρτης μνήμης δεν αντιγράφεται αμέσως αλλά υπάρχουν δείκτες κι από τις δύο διεργασίες γονιό και παιδί στις ίδιες θέσεις μνήμης όσο δε γίνεται `write` σε κάποια θέση μνήμης.

Ερώτηση 9

Οι φυσικές διευθύνσεις είναι διαφορετικές για τον πατέρα και το παιδί καθώς το παιδί έγραψε στον `private buffer`. Επειδή ο `buffer` είναι `private` για τον καθένα πρέπει να γίνει αντιγραφή σε άλλη θέση μνήμης, συνεπώς έχουμε διαφορετικές φυσικές διευθύνσεις (`copy on write`).

```
Step 9: Write to the private buffer from the child and repeat step 8. What happened?

parent pa=0x8dba2000
child pa=0xa623d000
```

Ερώτηση 10

Γράφοντας στον κοινό `buffer` από τη διεργασία παιδί βλέπουμε πως οι δύο φυσικές διευθύνσεις για τον πατέρα και το παιδί είναι ίδιες καθώς εφόσον είναι κοινός ο `buffer` δε χρειάζεται να αντιγραφεί σε άλλη θέση μνήμης.

```
Step 10: Write to the shared heap buffer (main) from child and get the physical address for both the parent and the child. What happened?

parent pa=0x56248000
child pa=0x56248000
```

Ερώτηση 11

Ακολουθεί ο χάρτης εικονικής μνήμης για τη διεργασία γονιό:

```
Virtual Memory Map of process [16725]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
01f5b000-01f7c000 rw-p 00000000 00:00 0                                  [heap]
7f53e0fc4000-7f53e1165000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1165000-7f53e1365000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1365000-7f53e1369000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1369000-7f53e136b000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e136b000-7f53e136f000 rw-p 00000000 00:00 0
7f53e136f000-7f53e1387000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1387000-7f53e1586000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1586000-7f53e1587000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1587000-7f53e1588000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1588000-7f53e158c000 rw-p 00000000 00:00 0
7f53e158c000-7f53e15ad000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e179f000-7f53e17a2000 rw-p 00000000 00:00 0
7f53e17a4000-7f53e17a5000 rw-p 00000000 00:00 0
7f53e17a5000-7f53e17a6000 rw-s 00000000 00:04 3992519                    /dev/zero (deleted)
7f53e17a6000-7f53e17a7000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7f53e17a7000-7f53e17ac000 rw-p 00000000 00:00 0
7f53e17ac000-7f53e17ad000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e17ad000-7f53e17ae000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e17ae000-7f53e17af000 rw-p 00000000 00:00 0
7ffcbe591000-7ffcbe5b2000 rw-p 00000000 00:00 0                          [stack]
7ffcbe5b7000-7ffcbe5ba000 r--p 00000000 00:00 0                          [vvar]
7ffcbe5ba000-7ffcbe5bc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Ακολουθεί ο χάρτης εικονικής μνήμης για τη διεργασία παιδί:

```
Virtual Memory Map of process [16726]:
00400000-00403000 r-xp 00000000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
00602000-00603000 rw-p 00002000 00:21 8549691                            /home/oslab/oslaba25/ex4/mmap
01f5b000-01f7c000 rw-p 00000000 00:00 0                                  [heap]
7f53e0fc4000-7f53e1165000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1165000-7f53e1365000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1365000-7f53e1369000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e1369000-7f53e136b000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f53e136b000-7f53e136f000 rw-p 00000000 00:00 0
7f53e136f000-7f53e1387000 r-xp 00000000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1387000-7f53e1586000 ---p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1586000-7f53e1587000 r--p 00017000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1587000-7f53e1588000 rw-p 00018000 08:01 6032223                    /lib/x86_64-linux-gnu/libpthread-2.19.so
7f53e1588000-7f53e158c000 rw-p 00000000 00:00 0
7f53e158c000-7f53e15ad000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e179f000-7f53e17a2000 rw-p 00000000 00:00 0
7f53e17a4000-7f53e17a5000 rw-p 00000000 00:00 0
7f53e17a5000-7f53e17a6000 r--s 00000000 00:04 3992519                    /dev/zero (deleted)
7f53e17a6000-7f53e17a7000 rw-p 00000000 00:21 8550076                    /home/oslab/oslaba25/ex4/file.txt
7f53e17a7000-7f53e17ac000 rw-p 00000000 00:00 0
7f53e17ac000-7f53e17ad000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e17ad000-7f53e17ae000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f53e17ae000-7f53e17af000 rw-p 00000000 00:00 0
7ffcbe591000-7ffcbe5b2000 rw-p 00000000 00:00 0                          [stack]
7ffcbe5b7000-7ffcbe5ba000 r--p 00000000 00:00 0                          [vvar]
7ffcbe5ba000-7ffcbe5bc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Όπως βλέπουμε στη δέκατη γραμμή από το τέλος στα `permissions` ο πατέρας έχει δικαιώματα `read` και `write`, ενώ το παιδί μόνο `read`. Η αλλαγή αυτή γίνεται μέσω της εντολής `mprotect()`.

Ερώτηση 12

Χρησιμοποιούμε την εντολή `munmap()` ώστε να αποδεσμεύσουμε κάθε έναν από τους `buffers` που θέλουμε να αποδεσμεύσουμε.

Άσκηση 2.1

Ακολουθεί ο πηγαίος κώδικας για την άσκηση 2.1:

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <pthread.h>
#include "mandel-lib.h"
#include <semaphore.h>
#include <errno.h>
#include <stdint.h>
#define perror_pthread(ret, msg) \
        do { errno = ret; perror(msg); } while (0)


#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/

/*
 * Output at the terminal is is x_chars wide by y_chars long
 */
sem_t *fst;
int y_chars = 50;
int x_chars = 90;
uint64_t va;
/*
```

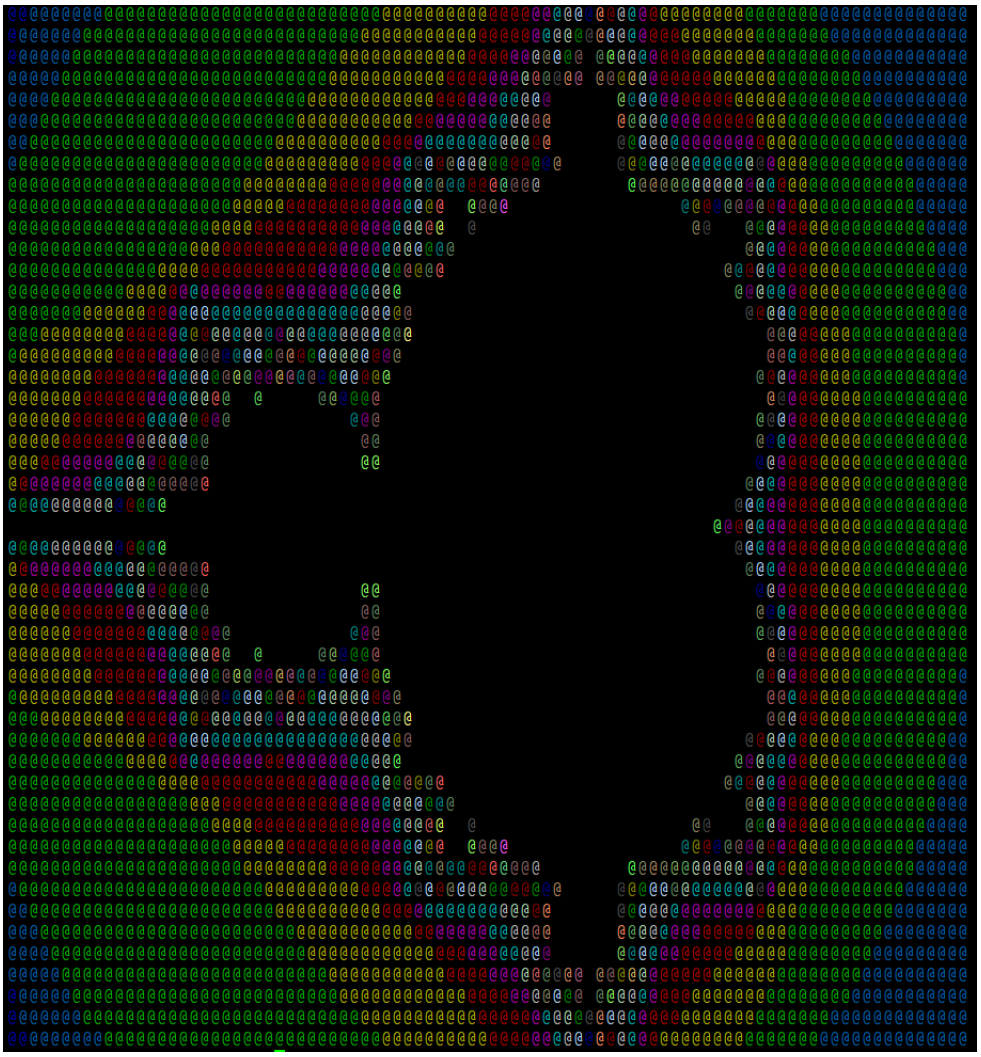
```
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax),
 lower right corner is (xmax, ymin)
 */
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

struct pair{
        int thrcount;
        int N;
};

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
        /*
         * x and y traverse the complex plane.
         */
        double x, y;

        int n;
        int val;

        /* Find out the y value corresponding to this line */
        y = ymax - ystep * line;

        /* and iterate for all points on this line */
        for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

                /* Compute the point's color value */
```

```
                val = mandel_iterations_at_point(x, y,
                MANDEL_MAX_ITERATION);
                if (val > 255)
                        val = 255;

                /* And store it in the color_val[] array */
                val = xterm_color(val);
                color_val[n] = val;

        }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
        int i;

        char point ='@';
        char newline='\n';

        for (i = 0; i < x_chars; i++) {
                /* Set the current color, then output the point */
                set_xterm_color(fd, color_val[i]);
                if (write(fd, &point, 1) != 1) {
                        perror("compute_and_output_mandel_line:
                        write point");
                        exit(1);
                }


        }
        /* Now that the line is done, output
        a newline character */
        if (write(fd, &newline, 1) != 1) {
                perror("compute_and_output_mandel_line:
                write newline");
                exit(1);
```

```
        }
}

void* compute_and_output_mandel_line(void *arg)
{
        int fd=1;
        volatile struct pair *pair = arg;
        int thrcount = pair->thrcount;
        int N = pair->N;
        int i;
        for(i=thrcount;i<y_chars;i+=N){
                /*
                * A temporary array, used
                to hold color values for the line being drawn
                 */
                int color_val[x_chars];
                compute_mandel_line(i, color_val);
                sem_wait(fst+thrcount);
                output_mandel_line(fd, color_val);
                if(thrcount==N-1){
                        sem_post(fst);
                }
                else{
                        sem_post(fst+thrcount+1);
                }
        }
        exit(12);
}

int main(int argc, char *argv[])
{
        int i,N;

        int ret;
        int status;
        N=atoi(argv[1]);

        struct pair pair[N];

        pid_t pr[N];
```

```
        va=mmap(NULL, N*sizeof(sem_t), PROT_READ|PROT_WRITE,
        MAP_SHARED|MAP_ANONYMOUS, -1, 0);
        sem_t *ptr=va;
        fst =va;
        for(i = 0 ; i<N; i++) {
                sem_init(ptr, 1, 0);
                ptr++;
        }


        sem_init (fst, 1, 1);

        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;

        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor
         '1', i.e., standard output.
         */
        for(i=0;i<N;i++){
                pair[i].thrcount=i;
                pair[i].N=N;
                pr[i]=fork();
                if (pr[i]==0) {

                        compute_and_output_mandel_line(&pair[i]);
                }

        }

        for(i=0; i<N; i++)
                pr[i]=wait(&status);

        reset_xterm_color(1);
        return 0;
}
```

Εκτελώντας τον κώδικα με τέσσερεις διεργασίες παίρνουμε το εξής αποτέλεσμα:



Ερώτηση 1

Η δημιουργία, η επικοινωνία και ο τερματισμός διεργασιών είναι πιο χρονοβόρες διαδικασίες από τη δημιουργία, την επικοινωνία και τον τερματισμό των νημάτων. Επίσης οι διεργασίες επιβαρύνουν πιο πολύ το σύστημα από ότι τα νήματα, αφού καταναλώνουν περισσότερους πόρους του συστήματος. Τέλος, τα νήματα έχουν ούτως ή άλλως κοινή μνήμη, ενώ στις διεργασίες πρέπει να δημιουργήσουμε εμείς κοινό χώρο μνήμης.

Έτσι, καταλήγουμε στο συμπέρασμα πως η βέλτιστη υλοποίηση αυτού του προβλήματος είναι με νήματα.

Άσκηση 2.2

Ακολουθεί ο πηγαίος κώδικας για την άσκηση 2.2.

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set
 on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <pthread.h>
#include "mandel-lib.h"
#include <semaphore.h>
#include <errno.h>
#include <stdint.h>
#define perror_pthread(ret, msg) \
        do { errno = ret; perror(msg); } while (0)


#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/

/*
 * Output at the terminal is is x_chars
 wide by y_chars long
 */
int *fst;
int y_chars = 50;
int x_chars = 90;
```

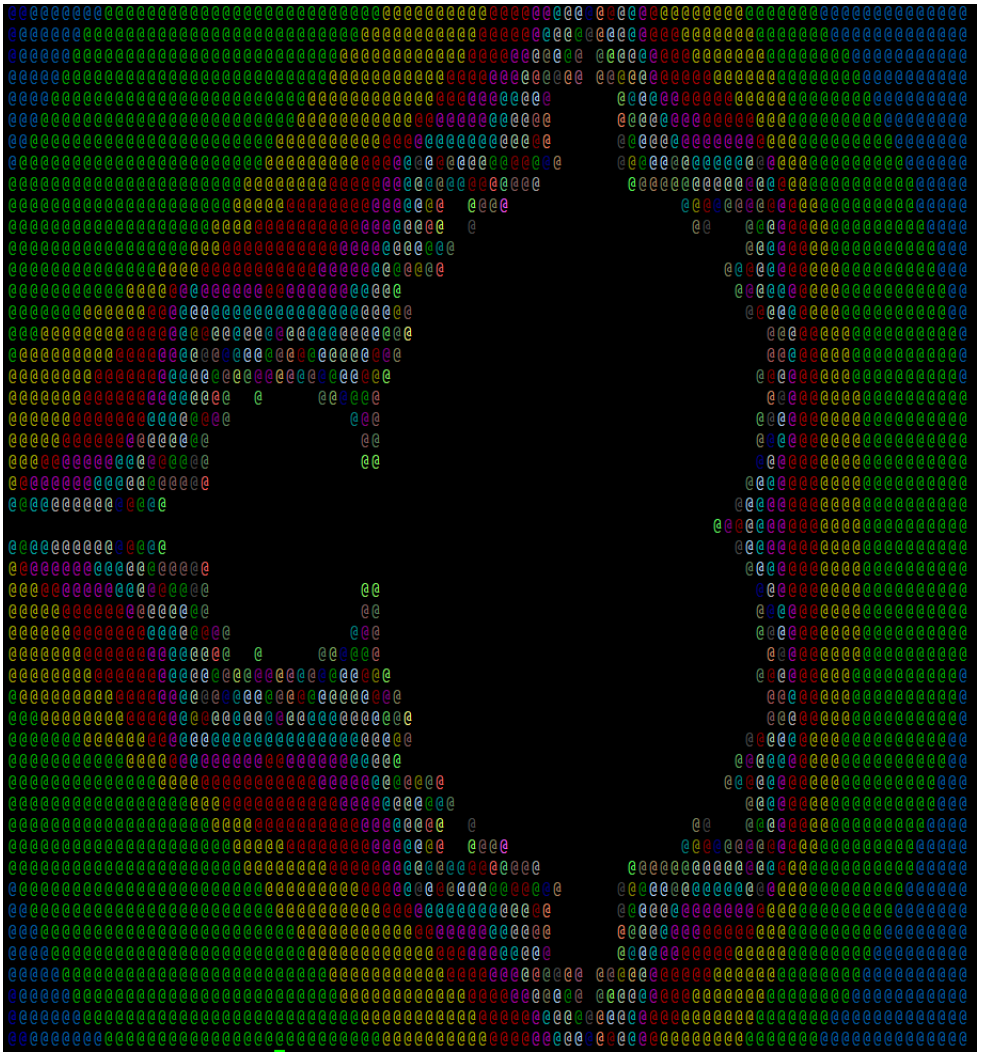
```
uint64_t va;
/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax),
 lower right corner is (xmax, ymin)
 */
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

struct pair{
        int thrcount;
        int N;
};

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
        /*
         * x and y traverse the complex plane.
         */
        double x, y;

        int n;
        int val;

        /* Find out the y value corresponding to this line */
        y = ymax - ystep * line;

        /* and iterate for all points on this line */
        for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
```

```
                /* Compute the point's color value */
                val = mandel_iterations_at_point(x, y,
                MANDEL_MAX_ITERATION);
                if (val > 255)
                        val = 255;

                /* And store it in the color_val[] array */
                val = xterm_color(val);
                color_val[n] = val;

        }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
        int i;

        char point ='@';
        char newline='\n';

        for (i = 0; i < x_chars; i++) {
                /* Set the current color,
                then output the point */
                set_xterm_color(fd, color_val[i]);

                if (write(fd, &point, 1) != 1){
                        perror("compute_and_output_mandel_line:
                        write point");
                        exit(1);
                }


        }
        /* Now that the line is done, output
        a newline character */
```

```
        if (write(fd, &newline, 1) != 1) {
                perror("compute_and_output_mandel_line:
                write newline");
                exit(1);
        }
}

void* compute_and_output_mandel_line(void *arg)
{
        volatile struct pair *pair = arg;
        int thrcount = pair->thrcount;
        int N = pair->N;
        int i,j;
        for(i=thrcount;i<y_chars;i+=N){

                /*
                * A temporary array, used to hold
                color values for the line being drawn
                 */

                int color_val[x_chars];


                compute_mandel_line(i, color_val);
                for(j=0;j<x_chars;j++)
                        *(fst + i*x_chars + j)=color_val[j];

        }
        exit(12);
}

int main(int argc, char *argv[])
{
        int i,N;
        int fd;
        int ret;
        int status;
        N=atoi(argv[1]);

        struct pair pair[N];
```

```
        pid_t pr[i];

        va=mmap(NULL, x_chars*y_chars*sizeof(int),
        PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
        fst = va;
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;

        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor
         '1', i.e., standard output.
         */
        for(i=0;i<N;i++){
                pair[i].thrcount=i;
                pair[i].N=N;
                pr[i]=fork();
                if (pr[i]==0) {

                        compute_and_output_mandel_line(&pair[i]);
                }

        }

        for(i=0; i<N; i++)
                pr[i]=wait(&status);

        fd=1;

        for(i=0; i<y_chars;i++)
                output_mandel_line(fd,fst + i*x_chars);

        reset_xterm_color(1);
        return 0;
}
```

Εκτελώντας τον κώδικα με τέσσερεις διεργασίες παίρνουμε το εξής αποτέλεσμα:



Ερώτηση 1

Ο συγχρονισμός σε αυτή την υλοποίηση επιτυγχάνεται με τη χρήση ενός `buffer` `y_chars*x_chars` όπου η διεργασία `i` θα να συμπληρώσει τους κωδικούς χρωμάτων στις γραμμές `i, i+N, i+2N` κοκ. Όταν συμπληρωθούν όλες οι γραμμές η διεργασία γονιός θα διαβάσει τον `buffer` και θα τυπώσει το `output`.

Ερώτηση 2

Εάν ο `buffer` είχε διαστάσεις `N_procs*x_chars` θα έπρεπε η διεργασία πατέρας ανά κάθε `N_procs` γραμμές να τυπώνει το `output` και στη συνέχεια να υπολογίζονται

οι κωδικοί χρωμάτων των επόμενων N γραμμών κοκ.